VŠB TECHNICKÁ UNIVERZITA OSTRAVA
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Hudební aplikace pro Garmin Connect IQ

Music App for Garmin Connect IQ

Ivo Pešák

Bakalářská práce

Vedoucí práce: Ing. Jan Janoušek

Ostrava, 2021

Abstrakt

Cílem této práce je navrhnout a implementovat aplikaci na chytré hodinky Garmin, která bude fungovat jako hudební přehrávač a aplikaci webovou, která bude sloužit ke spravování hudebního obsahu pro hodinky. Pro webovou aplikaci byli použity technologie .NET, C# a frontenfový framework Angular. Aplikace v hodinkách je implementována pomocí Connect IQ a jazyku Monkey C. Výsledkem této práce je funkční webová aplikace, kterou je možné propojit se službami Dropbox nebo Google Drive, kde je nahrán hudební obsah uživatele a hudební přehráváč na hodinky, umožňující tento obsah přehrát.

Klíčová slova

Bakalářská práce; Angular; .NET; C#; Connect IQ

Abstract

The goal of this bachelor's thesis is to design and implement two applications. One for Garmin smartwatch to play the music content of the user and one, which is a web application, to manage the content of the user. The web application is built by .NET and C# technology and front end framework Angular. The smartwatch application is built by the Connect IQ and Monkey C technology. The output of this bachelor's thesis is working web application which can be connected to cloud storage Dropbox or Google Drive. The music content of the user is stored in these storages and user can manage them through the web application. The smartwatch application then can play the user's content.

Keywords

Bachelor's thesis; Angular; .NET; C#; Connect IQ

Poděkování

Rád bych poděkoval panu Ing. Janouškovi za jeho vždy vstřícný přístup na konzultacích, při řešení problémů a za množství rad, které při mi při psaní práce poskytl.

Obsah

Seznam použitých zkratek a symbolů

HTML	–	Hypertext Markup Language
CSS	–	Cascading Style Sheets
CLR	–	Common Language Runtime
CIL	–	Common Intermediate Language
URL	–	Uniform Resource Locator
UI	–	User Interface
JWT	–	JSON Web Tokens
IDE	–	Integrated Development Environment
HTTP	–	Hypertext Transfer Protocol
JSON	–	JavaScript Object Notation
SDK	–	Software Development Kit
API	–	Application Programming Interface

Seznam obrázků

Kapitola 1

Úvod

V dnešní době se stále více využívají mobilní telefony. Od dob kdy telefony sloužili pouze k hovorům a SMS zprávám se technicky pokročilo k zařízením, které jsou téměř schopné všeho, čeho je schopný stolní počítač. Ovšem pokrok zašel ještě dál a kromě telefonů již existují i různé chytré doplňky, které, ať už jsou propojeny s telefonem, či nikoliv, přinášejí další řadu užitečných funkcí. Mezi tyto doplňky se řadí i chytré hodinky tzv. Smart Watch. Výrobců těchto hodinek je několik a i když se liší designem, tak jejich modely nabízejí, co se týče funkčnosti, podobné služby. Tyto služby mají mezi sebou řadu variací, od základního sledování času přes čtení SMS zpráv až po měření srdečního tepu. Další obvyklou funkcionalitou těchto chytrých hodinek je poslouchání hudby pomocí aplikací.

Cílem této práce je navrhnout a implementovat takovou hudební aplikaci. Dále také aplikací webovou, která bude sloužit jako vstupní bod pro nové uživatele a bude umožňovat správu hudebního obsahu uživatele. K tomuto účelu byly zvoleny technologie Angular, .NET a Connect IQ.

Tato práce je rozdělena na 3 části. První část se zabývá platformou Connect IQ a programovacím jazykem Monkey C a přibližuje vlastnosti tohoto jazyka. Druhá se věnuje návrhu aplikace a to jak aplikace na chytré hodinky, tak aplikace webové a rozebírá vlastnosti jednotlivých technologií, které byly použity pro implementaci. Poslední část je věnována implementaci aplikace a popisuje některé detaily obou aplikací.

Kapitola 2

Connect IQ

Connect IQ je vývojářská platforma pro aplikace, běžící na chytrých hodinkách značky Garmin. K vývoji samotných aplikací se využívá Connect IQ SDK a programovací jazyk Monkey C. Connect IQ nabízí vývoj aplikací typu Watch Face, Data Field, Widget, Device App a Audio Content Provider [1].

- Watch Face – Aplikace se zobrazuje jako domovská stránka pro hodinky. Má limitované možnosti vstupu od uživatele
- Data Field – Aplikace má přístup k datům, které se zobrazují v Garmin activities a umožňuje je modifikovat pro lepší prezentaci uživateli.
- Widget – Malá lehká aplikace, která uživateli poskytuje nějaká jednoduchá data. Má limitované možnosti vstupu od uživatele a po nečinnosti se přesune do pozadí.
- Device App – Plnohodnotná aplikace, která nemá žádné omezení na vstupy od uživatele a životnost má do doby, než ji uživatel sám ukončí.
- Audio Content Provider – Aplikace sloužící jako hudební přehrávač.

2.1 Monkey C

Jedná se o programovací jazyk, používaný pro vývoj aplikací. Je to jazyk objektově orientovaný. Kód v Monkey C je kompilován do kódu bytového, který je pak dále, podobně jako například JAVA, interpretován virtuálním strojem. Jazyk je dynamicky typovaný, přesněji uplatňuje Duck typing - kachní typování. Kachní typování využívá pravidla „Pokud vidím ptáka, který chodí jako kachna, plave jako kachna a kváká jako kachna, tak o tomto ptáku tvrdím, že je to kachna.“ [2]. Vedle základních datových typů Boolean, Char, String, Number, Long, Float, a Double umožnůje ještě používat Array a Dictionary, nicméně výše zmíněné základní datové typy jsou také objekty.

Proměnné můžeme definovat pomocí klíčového slova var. Pro přístup k aktuální instanci objektu se používá klíčové slovo self.

```
var slovnik = {
    "klic1" => "hodnota1",
    "klic2" => "hodnota2"
};
```

Listing 2.1: Ukázka slovníku v Monkey C

Podobně jako v jazyce JAVA jsou všechny objekty tvořeny v haldě. Paměť je čištěna virtuálním strojem pomocí počítání odkazů. To znamená, že ve chvíli kdy na objekt už nikdo nemá žádný odkaz, tak je jeho paměť uvolněna. Funkce v Monkey C jsou definovány, podobně jako v jazyce JavaScript, pomocí klíčového slova function a neuvádí se žádný návratový typ, ani v případě že funkce má něco vracet. Pouze ve funkci se hodnota vrátí pomocí klíčového slova return. Mezi další vlastnosti jazyka patří existence tzv. symbolů. Symbol je v podstatě odlehčená konstanta, která nevyžaduje předešlou deklaraci. Označuje se dvojtečkou před názvem.

```
var s = :mujSymbol;
```

Listing 2.2: Ukázka symbolu v Monkey C

Některé jazykem již definované symboly můžou být použity k anotaci funkcí, metod nebo proměnných a dodávají jim další speciální vlastnosti. Některé tyto symboly a jejich vlastnosti jsou popsány následujícím seznamem.

- :debug – Kód s touto anotací je pouze v ladícím režimu a nebude obsažen v režimu vydání
- :release – Kód s touto anotací je pouze v režimu vydání a nebude obsažen v ladícím režimu
- :test – Tato anotace označuje kód, který patří k unit testům

Příklad použití pro symboly :debug a :release třeba je situace, kdy máme proměnnou, po které požadujeme, aby měla jinou hodnotu v době ladění a při vydání. Nebo potřebujeme jiné chování pro funkci, závisející na tom, v jakém režimu aplikace zrovna operuje.

```
(:debug) function getApiUrl() { // pouze v ladícím režimu
    return API_DEV;
}
(:release) function getApiUrl() { // pouze v režimu vydání
    return API_PROD;
```

```
}
```

Listing 2.3: Ukázka anotací v Monkey C

Výjimky se v Monkey C dají odchytnout podobně jako například v jazyce C# a to pomocí bloku try, catch a finally. [3]

Kapitola 3

Návrh

Při vývoji aplikace je vhodné si předem udělat její návrh. Způsobů jak tento návrh udělat je několik a každá firma nebo vývojář má svůj vlastní osvědčený postup. Někteří si tvoří UML diagramy a případy užití, jiní si vystačí s popisem aplikace a vlastně žádný velký návrh nedělají. Součástí tohoto procesu je také zvolení vhodných technologií, které budou při vývoji použity. V následujících kapitolách se zaměříme na technologie, které byly použity při vývoji aplikace.

3.1 Webová aplikace

Webová aplikace je aplikace, kterou uživatel nemusí sám instalovat, ale má k ni přístup přes internetový prohlížeč [4]. A právě protože prohlížeč má nainstalovaný snad každý osobní počítač, tak je potencionální dosah těchto aplikací ohromný. Webová aplikace navrhnuta v této bakalářské práci se dělí na tzv. frontend a backend.

3.1.1 Frontend

Frontendem by se v oblasti webových aplikací dala označit část aplikace, která běží v prohlížeči uživatele. Tedy ta část, která není na serveru, ale je blíže samotnému uživateli. Běžně to je část psaná v HTML, CSS a jazyce JavaScript. K psaní frontendové části aplikace už máme ovšem v dnešní době na výběr i různé frontendové frameworky nebo knihovny. Jedním z takových frameworků je angular a právě v něm je psaná část webové aplikace.

Angular je frontendový framework psaný v jazyce TypeScript, sloužící k vývoji jednostránkových webových aplikací a je vyvíjen společností Google. TypeScript je nadstavba pro JavaScript a doplňuje JavaScript o další prvky. Jeden z hlavních prvků, o které JavaScript rozšiřuje, je statické typování. Díky tomu se v TypeScript můžou předem deklarovat proměnné s nějakým datovým typem pro nějaký účel. Datové typy se píší za dvojtečku. Typy je možné například psát i k návratovým hodnotám funkcí, jak je zvykem v jiných jazycích. Existuje také univerzální datový typ any, se kterým je možné do proměnné uložit cokoliv, stejně jako v klasickém JavaScript a bez jakéhokoli

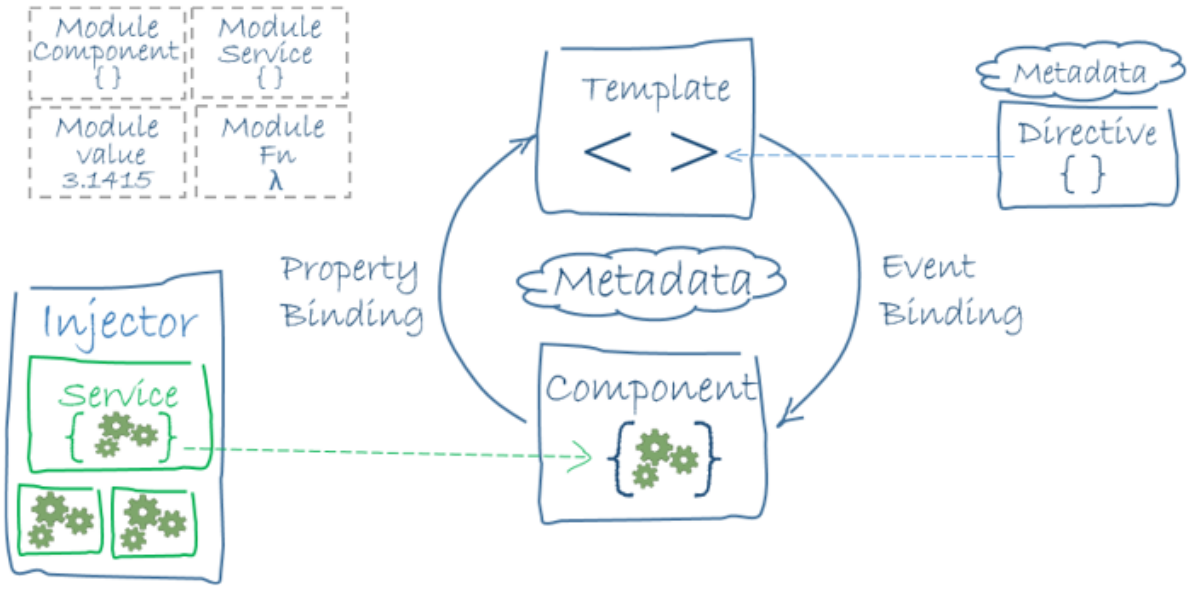
varování. Kód napsaný v TypeScript se ale nakonec stejně kompiluje zpátky do JavaScript a je jako JavaScript spouštěn. Takže každý JavaScript kód je také validním TypeScript kódem. Protože se nakonec stejně spouští JavaScript a ten nemá statické datové typy, ale má je dynamické, tak se datové typy definované v TypeScript dají přepsat typem jiným. Z toho důvodu statické typování slouží hlavně k psaní čistého kódu a je to pomocník pro programátora, aby byl kód udržitelný. Díky tomu je do proměnných ukládáno opravdu to, co by tam být uloženo mělo, za předpokladu že programátor dbá TypeScript varování.

```
let cislo: number = 5;
let retezec: string;
retezec = "slovo";
retezec = 5; // neni chyba, ale varování
let cokoli: any;
```

Listing 3.1: Ukázka proměnných v TypeScript

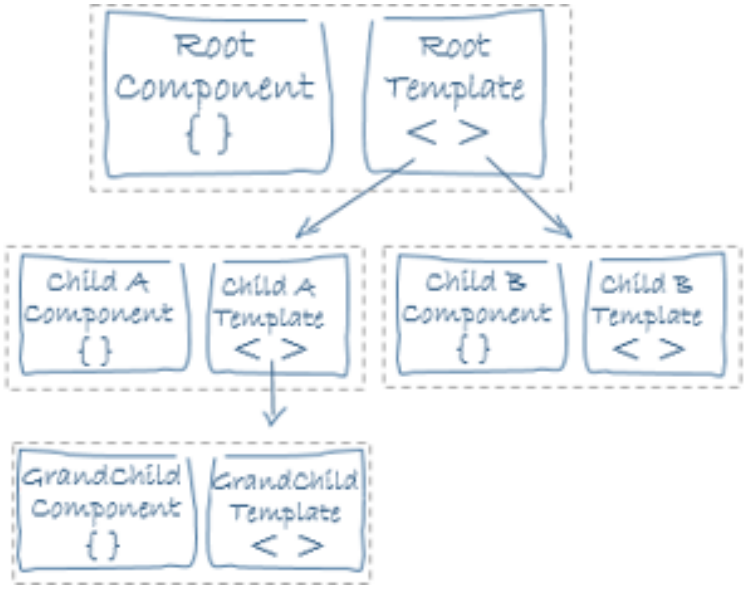
Pro další psaní čistého kódu TypeScript nabízí různá klíčová slova pro deklaraci proměnných. Kromě klasického var z JavaScript se také nabízí let a const. Rozdíl mezi nimi je ten, že hodnota proměnné definováné slovem const nemůže být upravena.

Architektura Angularu se zakládá na znovupoužitelných komponentách, které jsou dále organizovány do modulů a mohou využívat služeb. Komponenta je základní stavební blok aplikace, skládající se z HTML šablony a byznys logiky. Jedná se v podstatě o TypeScript třídu, označenou dekorátorem. Dekorátor je funkce, která modifikuje danou třídu (v tomto případe ji modifikuje jako komponentu). Angular má takových dekorátotů mnoho např. na označení modulu, nebo služeb a mnoho dalších. Dále pak komponenta musí patřit nějakému modulu. Modul shlukuje komponenty nebo služby a poskytuje tak balíček použitelný v aplikaci. [5]



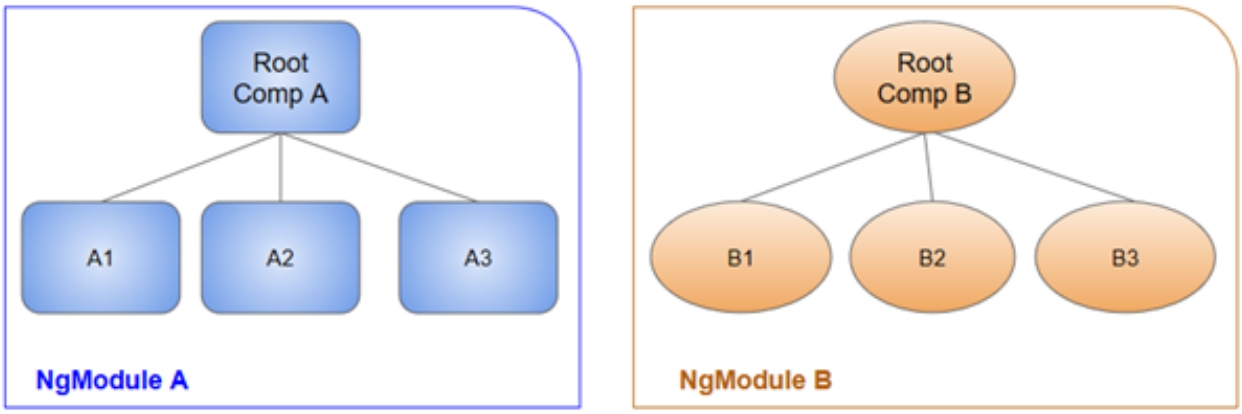
Obrázek 3.1: Struktura komponenty v Angularu [5]

Komponenty jsou znovupoužitelné a to jak celá komponenta, tak jenom její část. Není tedy problém použít stejnou HTML šablonu a CSS styly pro více komponent, či použít celou komponentu v aplikaci několikrát. Výhodou komponent je také jejich propojení s šablonou pomocí tzv. Two-way binding. Je to způsob provázání dat z modelu s šablonou takovým způsobem, že v případě změny dat v šabloně nebo modelu si druhá strana automaticky tyto data aktualizuje. Komponenty se také mohou vnořovat do sebe a tvořit stromovou strukturu.[6]



Obrázek 3.2: Větvení komponent v Angularu [6]

Moduly slouží ke shlukování komponent nebo služeb. Každá aplikace musí mít alespoň jeden takový modul. Výhoda více modulů je oddělení částí aplikace a jejich logiky dle komponent. Další výhoda spočívá v možnosti využití tzv. Lazy load. To je způsob načítání aplikace a sice takový, že se potřebná část aplikace (modul) načte až ve chvíli, kdy je potřeba. Bez využití tohoto způsobu by se celá aplikace musela načíst najednou, což by u větších aplikací mohlo způsobit problémy s výkonem. Ovšem při správném členění modulů a použití výše zmíněného způsobu, se toho riziko dá minimalizovat.[7]



Obrázek 3.3: Možná struktura dvou modulů v Angularu [7]

3.1.2 Backend

Pokud frontend je část aplikace, co běží v prohlížeči uživatele, tak backend bude pravý opak, tedy část aplikace, která běží někde na nějakém serveru a uživatel k ní a jejímu zdrojovému kódu nemá přístup. Pro backend, stejně jako pro frontend, máme mnoho možných technologií, ze kterých je možné vybírat např. Java, Python, Ruby atd. Dají se použít také backendové frameworky, jako je například Django jazyka Python. V backendové části webové aplikace je použita technologie .NET a přesněji ASP.NET Core.

.NET je open-source vývojářská platforma od firmy Microsoft, sloužící k vývoji aplikací různého typu. Pro tento vývoj lze použít programovací jazyky C#, F# a Visual Basic a lze cílit na platformy .NET Core, .NET Framework, Xamarin/Mono a .NET Standart [8]. Pro vývoj aplikací v této platformě existuje program Visual Studio, který je přímo k tomuto účelu určený. Část webové aplikace byla napsána právě v jazyce C#.

ASP.NET Core je open-source webový framework. Slouží k vytváření cloudových webových aplikací. Protože běží na technologii .NET Core, tak funguje na různých operačních systémech a procesorových architekturách. ASP.NET Core dále nabízí poddruhy aplikací, které se dají vytvořit. Jsou to například Razor Pages, ASP.NET Core MVC nebo gRPC service. Všechny tyto druhy ale stejně vznikly nějakou úpravou čistého ASP.NET Core projektu. Proto není problém vzít čistý

ASP.NET Core projekt a upravit ho podle našich požadavků, nebo upravit například MVC na projekt jiný.

C# je objektově orientovaný jazyk, vycházející z jazyků C a C++. Jde o jazyk silně typovaný a syntaxí se příliš neliší od svých rodičů. Aplikace psané v jazyce C#, nebo jiném jazyku z rodiny .NET, se kompilují do CIL kódu. Díky kompilaci do CIL kódu jsou .NET jazyky mezi sebou navzájem kompatibilní, tudíž není problém napsat část aplikace v C# a jinou třeba ve Visual Basic. Při spuštění se .NET aplikace spustí do virtuálního prostředí zvaného CLR, které ji dále spravuje. CLR právě dále převede CIL kód do kódu strojového pomocí Just-In-Time kompilace, což je kompilace, která se provádí až když je třeba daný úsek kódu provést. Mezi další činnosti, které CLR provádí, patří například správa paměti a zpracovávání výjimek.

C# rozeznává dva druhy datových typů: hodnotový typ a referenční typ. U proměnných hodnotového typu má každá proměnná svou vlastní kopii dat, zatímco u typu referenčního má každá proměnná odkaz (referenci) na tutéž hodnotu (tentýž objekt), takže v případě úpravy se hodnota upraví ve všech proměnných. Hodnotové typy se dále dělí primitivní typy, výčtové typy, struktury, nullable typy a tuple. Referenční typy se dále dělí třídy, rozhraní, pole a delegáty.

Třídy v C# můžou obsahovat i tzv. vlastnosti. Vypadají podobně jako atributy, ale ve skutečnosti jde o funkce. Možností zápisu je několik.

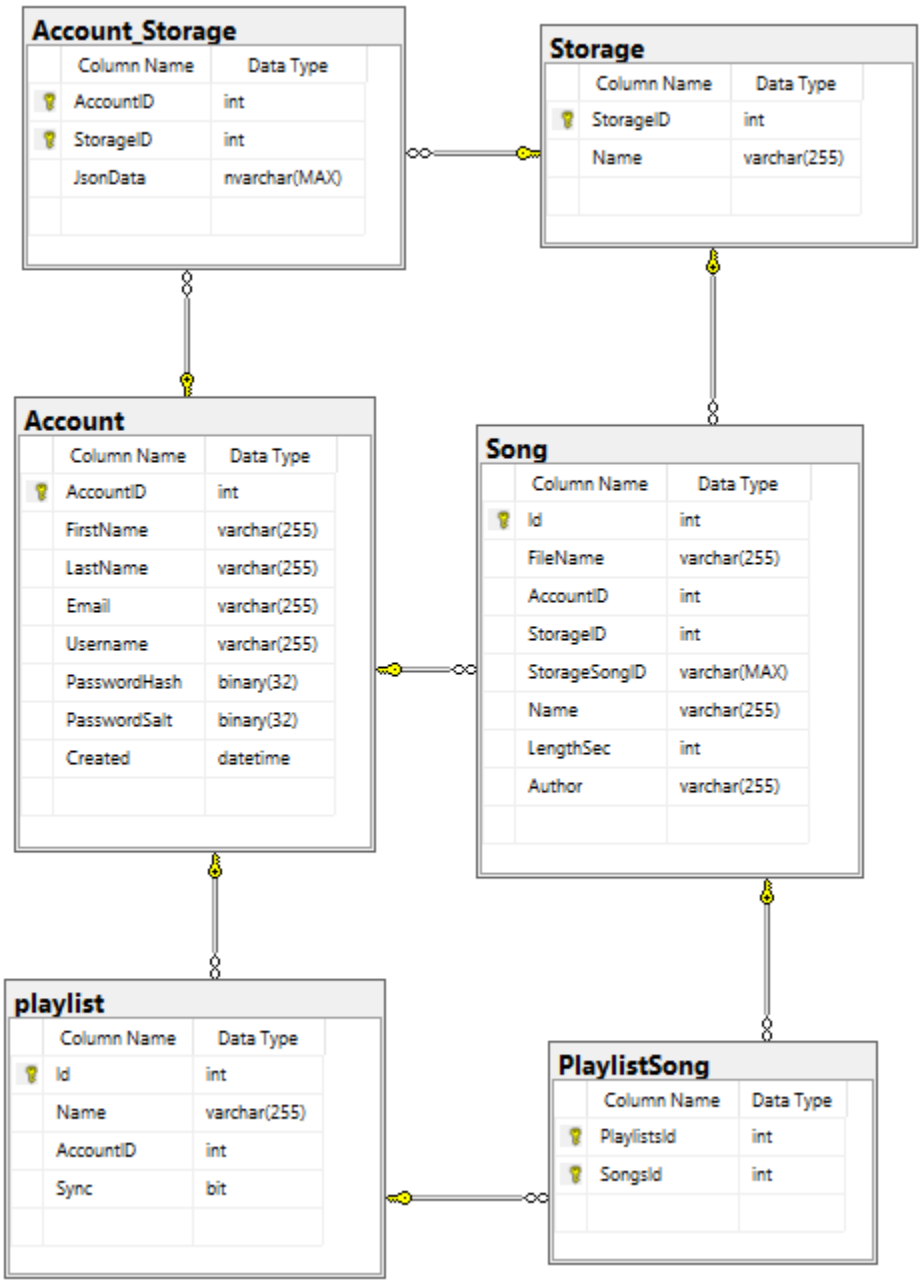
```
class Trida
{
    private int a;
    public int A // Toto je vlastnost
    {
        get
        {
            return a;
        }
        set
        {
            a = value;
        }
    }
    public int B { get; set; } // Toto je vlastnost
}
```

Listing 3.2: Ukázka vlastností v C#

V případě vlastnosti „A“ se vytvoří dvě funkce. První která bere jako parametr int a nastaví hodnotu privátní proměnné „a“ a druhá je bez parametru a vrací hodnotu privátní proměnné „a“.

Tento delší zápis se dá zkrátit v případě vlastnosti „B“. Ta ve finále udělá úplně to samé, akorát pro privátní proměnnou „b“, kterou si sama vytvoří. Vlastnosti se hodí pro tzv. gettery a settery, tedy funkce, které mají pouze nastavovat nebo vracet hodnotu nějaké proměnné. A protože jde o funkce, tak do tohoto přiřazení hodnoty, nebo jejího vrácení, může být zakomponována i nějaká logika. K vlastnostem se potom přistupuje ne jako k funkcím, ale jako k normálním atributům třídy. Ke klíčovým slovům get a set se dá také přidat modifikátor přístupu, který pak omezí přístup k dané vlastnosti. Dá se tak vytvořit například vlastnost s privátním setterem, ale veřejným getterem. Také se dá jedno z těchto klíčových slov vynechat úplně a vytvořit jak vlastnost, která je pouze getter nebo pouze setter. Je ale důležité si uvědomit že vlastnosti se nemusí použít pouze k získávání, nebo nastavování hodnot nějakého atributu. Jde přeci o funkci, takže je z ní možno získat například nějaký výpočet. Například ve třídě Kruh, by mohla být vlastnost Obsah, který by vracela vypočtený obsah. Tedy vlastnosti se neomezují pouze na hodnotu privátní proměnné se stejným názvem. Důvod proč používat pro takové účely vlastnosti a ne explicitně psané funkce, je ten, že vlastnosti se dají zapsat rychleji a jejich používání v kódu je jednodušší, protože se k nim přistupuje stejně jako k atributům. Nevypadá to, že se volá nějaká funkce. Toto je ovšem na preferencích programátora a někdo bude raději používat funkce, než vlastnosti.[9]

Jako uložiště dat pro webovou aplikaci slouží relační databáze, která obsahuje informace o uživatelích a jejich hudebním obsahu, ovšem už neobsahuje samotný hudení obsah. To by vedlo k ohromné velikosti databáze což je nežádoucí. Hudební obsah je místo toho ukládán na cloudové uložiště uživatele a v databázi se udržují pouze potřebné údaje k získání tohoto obsahu.



Obrázek 3.4: E-R model databáze

3.2 Chytré hodinky

Aplikace na chytrých hodinkách je postavena technologii Connect IQ a je napsána v jazyce Monkey C. Komunikuje s backendem a stará se o synchronizaci hudebního obsahu na hodinkách a o

jeho přehrávání. Jde o aplikaci typu Audio Content Provider. Aplikace operuje ve čtyřech hlavních módech a to přehrávání, konfigurace přehrávání, synchronizace a konfigurace synchronizace. Mód přehrávání slouží k přehrávání hudebního obsahu. V konfiguraci přehrávání si uživatel může vybrat který hudební obsah chce přehrát a ještě ovládací tlačítka pro přehrávací mód. Synchronizace synchronizuje hudební obsah s backendem a stahuje příslušné písničky do zařízení. Mód konfigurace synchronizace je pro účely přihlášení a odhlášení uživatele.[10]

Kapitola 4

Implementace

4.1 Webová aplikace

Následující kapitoly se věnují podrobnostem o implementaci webové aplikace. Jsou v nich také nastíněny některé problémy, které se během vývoje objevily a také jejich řešení.

4.1.1 Frontend

K vývoji frontendové části byl využit program Visual Studio Code. Jde o jednoduchý textový editor od společnosti Microsoft se schopností doinstalovat rozšíření s různou funkcionalitou. Díky této vlastnosti program ulehčuje vývoj v různých jazycích a frameworcích, pokud je k dispozici dané rozšíření např. na zvýraznění syntaxe, či doplňování útržků kódu. Tyto rozšíření umožňuje libovolně vypínat a zapínat, nebo je možnost nastavit si vybraná rozšíření jenom pro nějaký workspace. Obsahuje také vlastní panel pro správu verzí. Pokud tedy programátor nevyužívá nějaký externí program pro správu verzí, tak může využít tento zabudovaný. Ten ovšem nemusí nabízet tolik možností jako externí program přímo určený pro tento účel. Dále program disponuje integrovaným terminálem, pro zadávání příkazů bez nutnosti používat terminál externí. V praxi program neotevírá pouze jednotlivé soubory, ale pracuje nad celou námi určenou složkou s projektem a umožňuje tak rychlou a intuitivní navigaci mezi soubory a jejich editaci. Složka s frontendovou aplikací se nachází uvnitř aplikace backendové. Pro organizaci zdrojových kódů frontendové aplikace byla zvolena struktura, která je ve zjednodušené verzi popsána následujícím seznamem.

- app

  Adresář obsahující zdrojové kódy, komponenty a moduly.

  - config

    Konfigurační soubor obsahující konstanty z environmentu, které dále upravuje pro použití v aplikaci.

  - main
    * Komponenty
      Komponenty aplikace
    * Shared
      Adresář obsahující komponenty, služby a modely využívané v komponentách napříč celou aplikací.

- assets

  Adresář obsahující média.

- environment

  Adresář s konstantami, lišícími se podle toho zda je aplikace v produkčním módu nebo je ještě vyvíjena. Obvykle řetězce URL pro volání backendu.

Pro design stránky byl použit Angular Material což je knihovna UI komponent, takže není potřeba vymýšlet vlastní CSS design. Komponenty mají taky nadefinované nějaké výchozí chování, jako je například změna barvy po označení kurzorem a jsou responzivní, stejně jako celá aplikace.

Protože je webová aplikace určena pro více uživatelů a každý má přístup k nějakým soukromým datům, tak je třeba aby se uživatelé registrovali a přihlašovali. K tomu slouží přihlašovací formulář, který posílá údaje na backend. Přihlašovací logice je věnována část sekce Backend, tady nám stačí vědět, že po úspěšném přihlášení nám server vrátí zpátky tzv. JWT, který se uloží do Local Storage v prohlížečí, odkud je následně používán.

JWT je nějaká sekvence znaků s předem definovanou a zahašovanou strukturou Header.Payload.Signature.

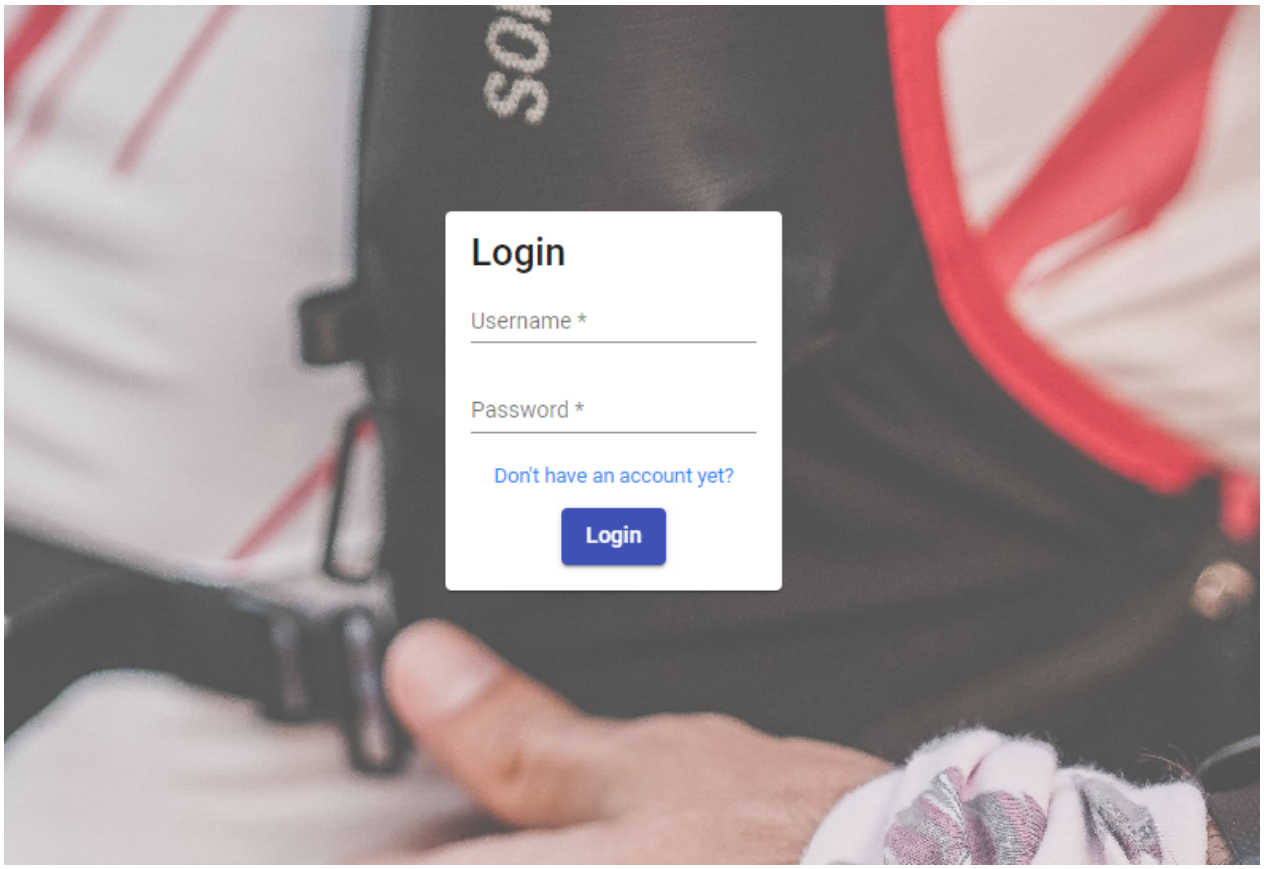
- Header – Hlavička nesoucí informace o jaký typ tokenu se jedná a podle jakého algoritmu je vytvořen

- Payload – Data obsahující nějaké předem definované informace tzv. Claims o tokenu, např. doba po kterou je platný, nebo kdo ho vydal. Dále také informace o entitě, které byl tento token přiřazen. Sekce Payload může být i prázdná, nicméně je doporučeno definovat nějaké Claims pro lepší bezpečnost aplikace.

- Signature – Podpis, který je vytvořen ze kombinací předchozích dvou částí, převedených do Base64Url, a tajným klíčem a poté jejich následným zahašováním.

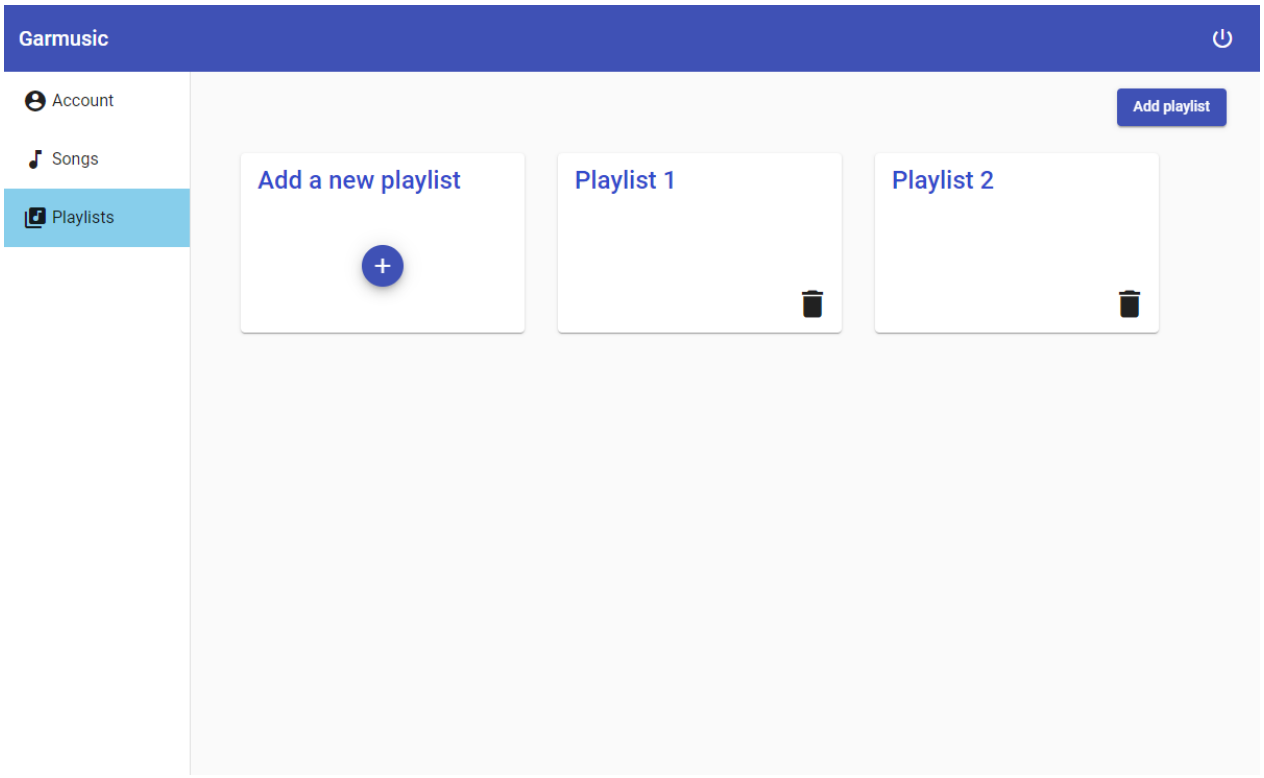
Takto vzniklý řetězec se následně zahašuje pomocí algoritmu, specifikovaného v části Header. JWT ovšem nezaručuje skrytost dat, pouze jejich pravost. Tedy data může kdokoli přečíst, ale pokud je pozmění tak zneplatní celý token[11].

JWT se posílá s každým požadavkem na server v hlavičce Authorization ve tvaru Authorization: Bearer <JWT> a server kontroluje jeho pravost.

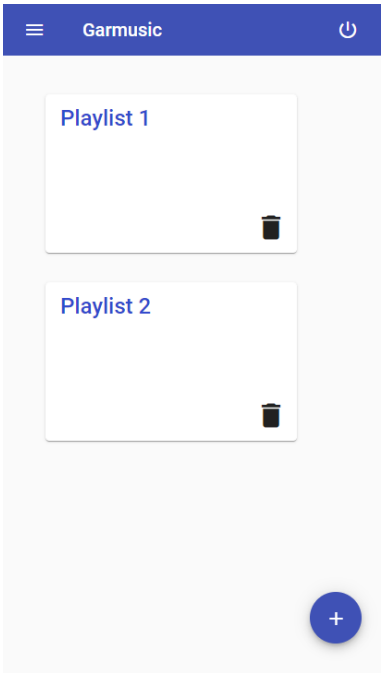


Obrázek 4.1: Přihlašovací stránka a formulář

Po přihlášení je uživatel přesměrován do hlavní části aplikace. Angular ovšem při přistupování na jednotlivé stránky kontroluje zda má JWT platnou životnost. Pokud nemá, tak uživatele vrátí na přihlašovací stránku a uživatel je nucen se znovu přihlásit, pro obdržení dalšího tokenu s obnovenou platností.



Obrázek 4.2: Stránka Playlists aplikace



Obrázek 4.3: Responzivní stránka Playlists aplikace

Aplikace má v levé sekci obrazovky menu pro navigaci mezi jednotlivými stránkami a uprostřed obsah stránky na které se nachází. Hlavní stránky jsou celkem tři.

- Account – Stránka pro spravování účtu a hlavně úložišť třetích stran.
- Songs – Stránka pro správu písniček. Uživatel zde může přidávat nebo odebírat písničky a modifikovat do jakých playlistů budou patřit.
- Playlists – Stránka pro správu playlistů. Umožňuje uživateli přidávat a odebírat playlisty a přidávat nebo odebírat do nich písničky.

Většina komponent využívá nějaké data, která jsou spjatá s uživatelem a tyto data načítá z backendu. Kdyby se data ukládala pouze v komponentách kde jsou používány, tak by se pokaždé, co nějaká jiná komponenta o tyto data požádá, musela načíst znova z backednu. Tudíž by se vytvořil nový HTTP požadavek. Protože jsou ale data po prvním získání uloženy ve službách, tak každá další komponenta dostane už načtená data z této služby a není nutné posílat nový HTTP požadavek. Je ale potřeba řešit situaci, když se data upraví. Pokud chceme omezit zbytečné HTTP požadavky na backend navíc, tak můžeme modifikovat přímo data ve službách. Toto je například používáno při přidání písničky do playlistu. Nebo pokud provádíme složitější změny a chceme si být jistí, že data jsou stejná jako v databázi, tak jednoduše data ve službě smažeme. Tímto způsobem si další komponenta, která bude data požadovat, stáhne aktuální data z backendu a tyto se opět uloží do služby a můžou z ní čerpat další komponenty.

Protože data ve službách jsou udržovány neustále, tak existují i v případě, kdy se uživatel odhlásí, ať to ručně, nebo když je odkázán na přihlášení z důvodu prošlého JWT. Kdyby se potom přihlásil jiný uživatel, tak uvidí stejná data, jako ten předchozí. Proto jsou data při každém odhlášení mazána.

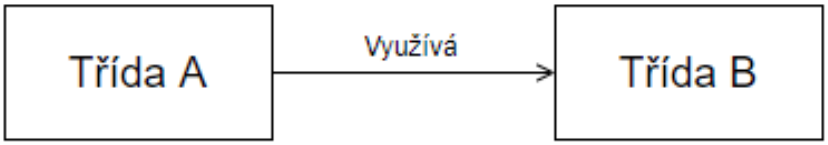
4.1.2 Backend

K vývoji backendové části bylo využito IDE Visual Studio, což je vývojové prostředí od společnosti Microsoft. Umožňuje vyvíjet různé typy aplikací v technologii .NET, ale není omezeno pouze tím. Dá se také třeba použít pouze jako textový editor. Pro organizaci zdrojových kódů byla zvolena struktura, která je ve zjednodušené verzi popsána následujícím seznamem.

- Garmusic
  - ClientApp – Adresář obsahující frontendovou aplikaci
  - Controllers – Adresář obsahující kontroléry, které naslouchají a zpracovávají HTTP požadavky.
  - Interfaces – Adresář obsahující rozhraní pro repozitáře a služby.

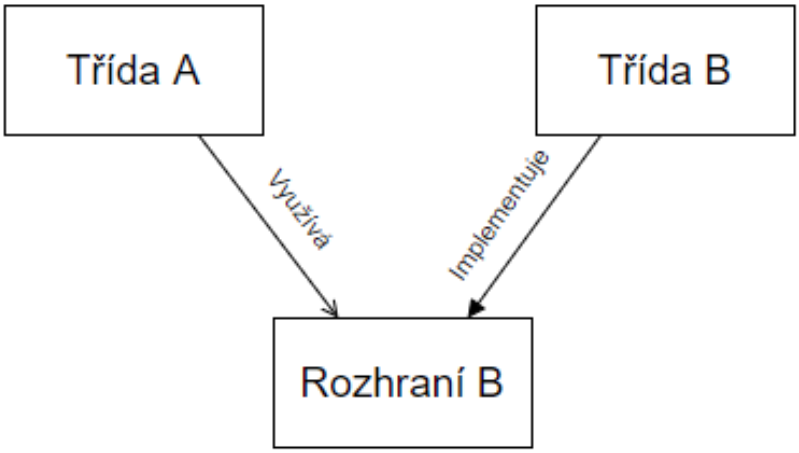
- Models – Adresář obsahující třídy-modely pro databázové a ostatní entity
- Repositories – Adresář obsahující třídy-repozitáře, které komunikují s úložišti dat
- Services – Adresář obsahující třídy-služby, které komunikují s repozitáři a služby které nastavují vlastnosti aplikace při jejím startu.
- Utilities – Adresář obsahující pomocné třídy, používané napříč aplikací.
- appsettings – Soubory s nastavením lišícími se podle toho zda je aplikace v produkčním módu nebo je ještě vyvíjena. Obvykle připojovací řetězce k databázi, URL adresa serveru a tajné řetězce pro hašování.

Rozhraní jsou použity pro tzv. Dependency inversion. Jde o princip obrácení závislostí tak, aby třída vyšší úrovně nebyla závislá na jiné třídě úrovně nižší, ale aby byly obě dvě závislé na nějakém rozhraní, tedy na nějaké abstrakci. Mějme následující obrázek.



Obrázek 4.4: Obrácení závislostí 1

Třída A je využívá třídu B a je tedy na ní závislá. V případě že se třída B nějak modifikuje, například změní své chování nebo definici, bude se muset změnit i třída A. Tento problém ovšem zmizí, pokud upravíme schéma dle obrázku níže.



Obrázek 4.5: Obrácení závislostí 2

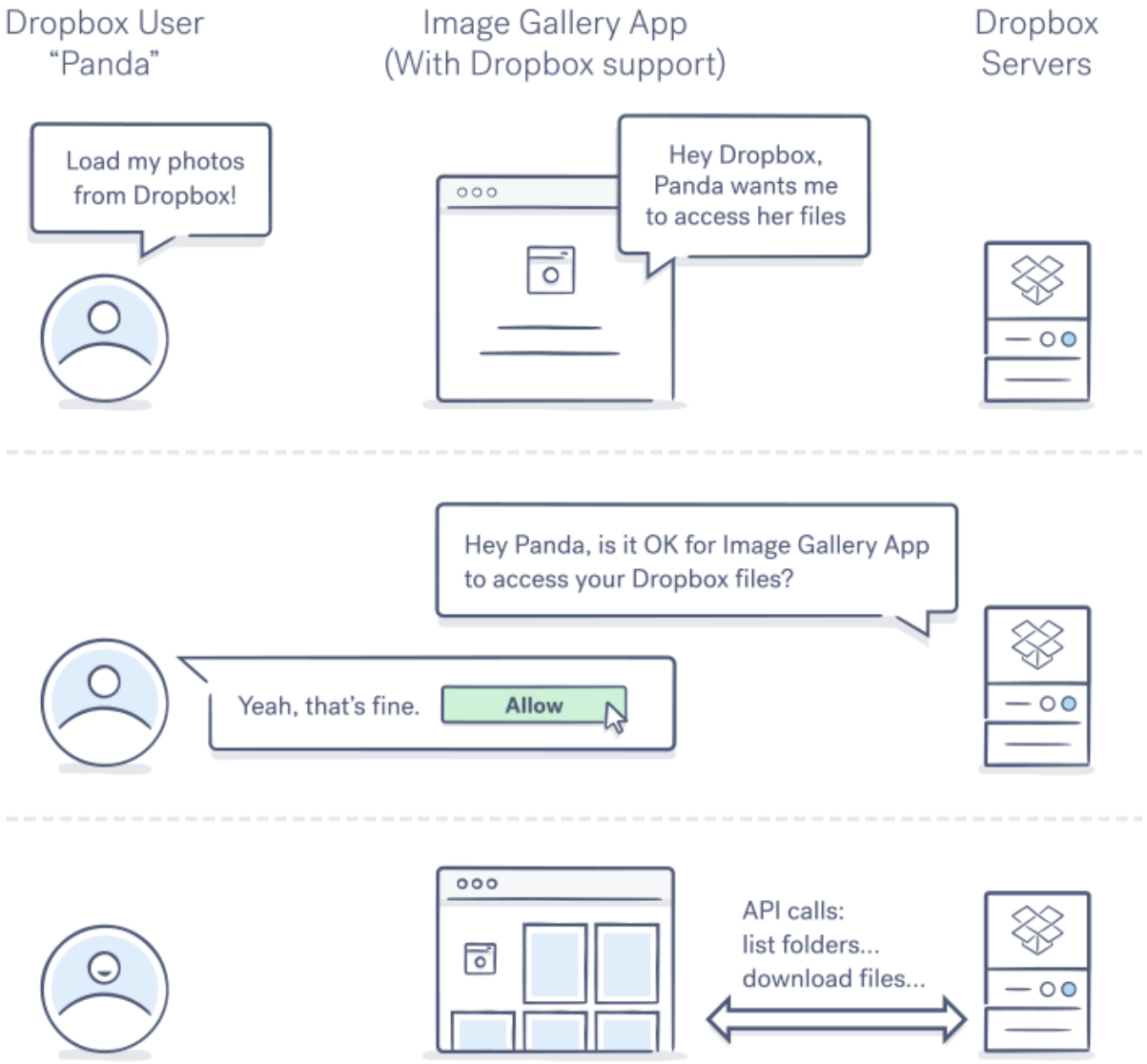
Třída A nyní využívá rozhraní B a to stejné rozhraní implementuje třída B, takže třída A se nemusí měnit ani v případě změny třídy B. Další výhoda tohoto principu je také možná změna implementace. Třídu B můžeme nahradit jakoukoliv jinou třídou, která implementuje rozhraní B a třída A se nemusí vůbec měnit [12].

Kontroléry jsou zabezpečeny před neověřenými HTTP požadavky. Toto zabezpečení je realizováno pomocí JWT v HTTP hlavičce požadavku. Aplikace .NET zkontroluje, zda je token platný a pokud ano, tak propustí požadavek do příslušného kontroléru, v opačném případe požadavek zamítne s chybovým kódem 401 Unauthorized. Toto zabezpečení ovšem některé metody v kontrolérech nemají. Pokud by tak nebylo, tak by uživatel musel mít JWT například při registraci nebo přihlašování, což samozřejmě není možné.

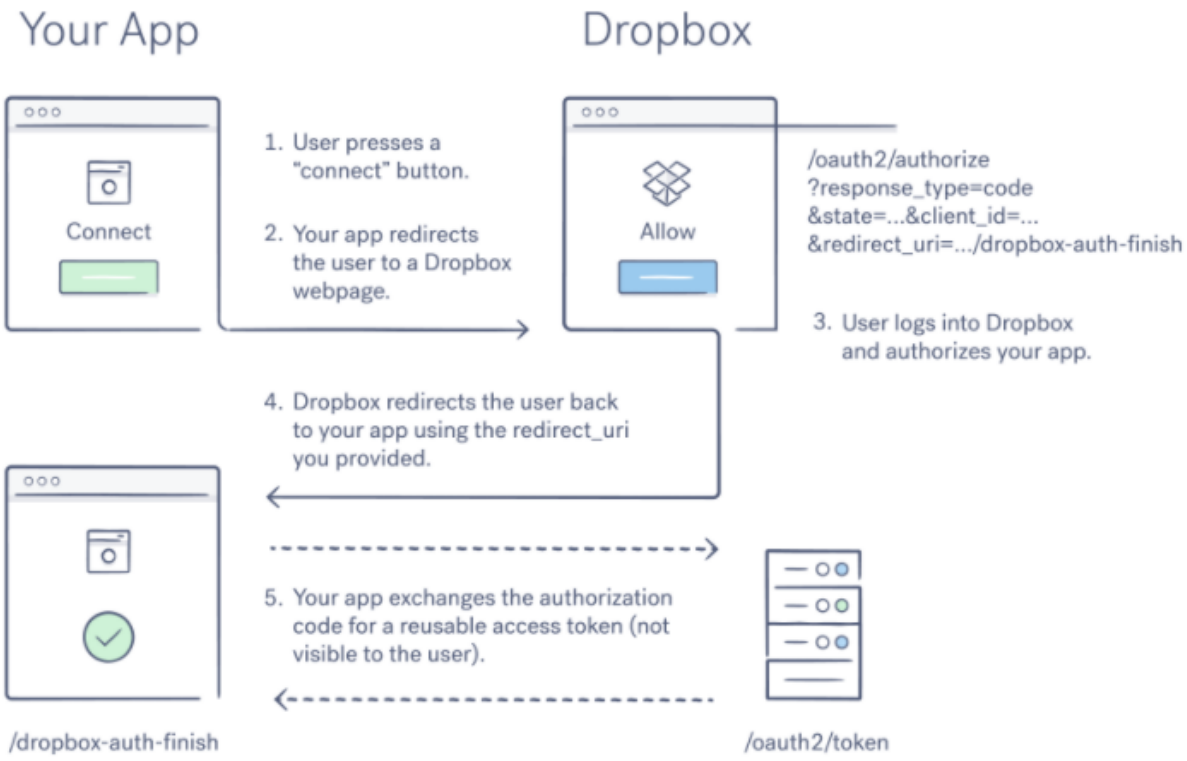
Registrace nového uživatele probíhá následujícím způsobem. Uživatel vyplní registrovací formulář na frontendové aplikaci a ta ho po validaci pošle na backend. Kontrolér data převezme a zkontroluje, jestli jsou údaje jako emailová adresa unikátní vůči databázi. Když tomu tak není, tak vrátí chybovou hlášku o této skutečnosti. Pokud jsou tyto údaje v pořádku, tak se připraví k uložení do databáze. Z bezpečnostních důvodu nechceme v databázi ukládat heslo uživatele. Jestli by došlo útoku na data a útočník by získal hesla, bylo by to enormní bezpečnostní riziko. Místo toho se heslo hašuje, což útočníkovi znesnadní práci, i kdyby se k datům dostal. Pokud by však nějaké heslo rozluštil, tak by pravděpodobně přišel i na ostatní hesla, které mají stejný zahašovaný tvar. Proto se pro ještě větší bezpečnost k heslům přidává tzv. kryptografická sůl. Jde o sekvenci náhodně vygenerovaných znaků, které jsou před hašováním přidání k heslu. Můžou být dány na začátek, konec, nebo nějakým způsobem rozmístěny mezi heslo samotné. Tímto způsobem budou dvě stejné hesla mít jiný zahašovaný tvar, za předpokladu že budou mít jinou sůl. Hašované heslo se společně se solí a dalšími údaji uloží do databáze a uživatel se může pod novým účtem přihlásit.

Přihlášení využívá onu sůl pro ověření přihlašovacích údajů. Podle přihlašovacího jména které uživatel zadá při přihlašování se z databáze získá sůl. Ta se přidá k zadanému heslu a zahašovaný výstup se porovná s daty v databázi. Pokud se přihlašovací jméno, nebo heslo neschoduje, tak je tato skutečnost zdělena uživateli. Pokud se shodují, tak je uživatel úspěšně ověřen a je vytvořen JWT, který je poté vrácen uživateli pro budoucí HTTP požadavky a navigaci na stránce.

Webová Aplikace využívá cloudových úložišť třetích stran pro ukládání hudebního obsahu. Jedno z takových úložišť, které aplikace podporuje, je Dropbox. Aby webová aplikace mohla úspěšně s tímto úložištěm komunikovat, programátor musí vytvořit Dropbox aplikaci ve Dropbox vývojářské konzoli. Tam nastaví potřebné údaje o jeho webové aplikaci, jako je URL adresa aplikace a vlastnosti tokenu, který je používán pro ověření komunikace. Pro přístup k úložišti uživatele, musí uživatel nejdříve autorizovat Dropbox aplikaci, přes kterou komunikuje aplikace webová. Po autorizaci obdrží webová aplikace autorizační token, který si s Dropboxem vymění za přístupový token, který má životnost definovanou, podle nastavení aplikace. [13]



Obrázek 4.6: Dropbox autorizace z pohledu uživatele [13]



Obrázek 4.7: Dropbox autorizace z pohledu programátora [13]

Aplikace má nastavenou životnost tokenu na nekonečnou. Tedy token bude stále platit. Tento token pak ukládá do databáze a nemusí tedy docházet k žádnému obnovování.

Když uživatel úspěšně autorizuje aplikaci a token je uložen do databáze, tak nastává proces uložení dat o hudebním obsahu z Dropboxu do databáze. Pokud by nebyla data v databázi, ale pokaždé by se data četla z Dropboxu, tak by to negativně ovlivnilo rychlost programu, kvůli dlouhým HTTP požadavkům.

Algoritmus 1: Prvotní uložení hudebních dat do databáze

```
Input: ID: Id uživatele
entita := NajdiUzivateleVDatabazi(ID);
if entita nenaleza then return;
dbxConnToken := PrectiToken(entita);
spojeni := DbxSpojeni(dbxConnToken);
soubory := spojeni.NactiSoubory();
UlozNovyKurzor(soubory.kurzor);
foreach soubor in soubory do
    if soubor není mp3 then continue;
    if soubor byl smazán then
        SmazSouborZDatabaze(soubor);
    else
        PridejSouborDoDatabaze(soubor);
    end
end
```

Dropbox navíc nabízí tzv. webhooky. Jde o funkci poslat požadavek na programátorem předem určenou adresu, když dojde ke změně na cloudovém disku uživatele. Spolu s požadavkem je poslána i informace o tom, u kterých uživatelů došlo ke změně. Webová aplikace je takto schopna aktivně reflektovat tyto změny a načítat nebo mazat data o souborech. Proto se ukládá i kurzor, který obdržíme při načtení dat a používá se při budoucím stahování dat. Pokud je použit, tak nám Dropbox vrátí seznam souborů od doby, kdy byl použitý kurzor vydán a jelikož kurzor ukládáme po každém načtení, tak je to doba od posledního načtení.

Aplikace funguje podobně i s úložištěm Google Drive. Programátor musí vytvořit aplikaci v Google vývojářské konzoli a vyplnit potřebná data, jako je například URL adresa aplikace. Z konzole pak programátor dostane tzv. Credentials, což jsou data, které se využívají k identifikaci a autentikaci aplikace při komunikaci s Google Api.

Při propojování webové aplikace ke Google Drive uživatele pak aplikace vygeneruje URL pomocí Credentials, na kterou přesměruje uživatele. Jde o autorizační stránku od Google, kde se uživatel přihlásí pod svým Google Drive účtem. Když odsouhlasí přístup, tak Google přesměruje uživatele na URL specifikovanou ve vygenerované URL. Stejně jako u postupu s Dropbox úložištěm je do URL po přesměrování zapsaný kód, který se potom použije k získání přístupového tokenu. Google ale nenabízí neomezenou životnost tokenu, tak je třeba využívat způsobu obnovování pomocí tzv. refresh tokenu. Ten je uložen v databázi a je používán k získání tokenu, který má životnost omezenou a je používán pro přístup k Google Drive.

Google Drive standardně zapisuje získané tokeny do JSON souboru, ze kterého pak také čte. Pokud mají být tokeny zapisovány jinam nebo s jinou logikou, musí se vytvořit třída, která im-

plementuje rozhraní IDataStore a tato třída musí být předána jako parametr do funkce, který na backendu autorizuje webovou aplikaci pro používání Google Drive uživatele.

Google Drive nabízí také webhooky stejně jako Dropbox. Rozdíl je ale ten, že u Dropboxu se posílají automaticky na předem definovanou adresu, zatímco u Google Drive je třeba poslat požadavek s informací o kterých změnách nás má informovat. V případě této aplikace to jsou změny v celém úložišti, ale nabízí se také možnost sledovat změny např. u jednoho konkrétního souboru. K požadavku doplníme identifikátor, který je uložen do databáze pro identifikaci uživatele, ke kterému změny patří. Ovšem požadavek má na rozdíl od Dropbox omezenou životnost a je třeba ho obnovovat. K obnovení dochází kdykoliv je tento webhook úspěšně obdržen, když je to poprvé co si uživatel přidává do aplikace Google Drive nebo když se uživatel přihlásí. Není ale způsob jak doslova obnovit tento požadavek, musí se proto poslat nový. Pokud by se jenom posílaly nové, tak by aplikace potom dostávala více oznámení k jedné změně, proto se musí staré požadavky zrušit a potom poslat nový. Tímto způsobem bude aktivní pouze jeden požadavek na sledování změn v úložišti uživatele.

Dropbox i Google Drive poskytuji svoje API pro technologii .NET, díky kterým je usnadněn vývoj aplikací, které tyto služby využívají. Bez těchto API by se musely sestavovat HTTP požadavky, které mohou být mnohdy složité a se spoustou parametrů. S API nám ale stačí zavolat příslušnou metodu, do ní vložit parametry a tato metoda pak sama vytvoří a pošle HTTP požadavek. Tímto způsobem pracuje backend webové aplikace, tedy neposílá ručně vytvořené HTTP požadavky, ale využívá API od služeb úložišť.

Při nahrávání písniček na úložiště se nejdříve pošle soubor z frontendu na backend společně s informací, na jaké úložiště se má píseň nahrát. Podle této informace se spustí příslušná metoda. Metody pro obě úložiště fungují na podobném principu. Vytvoří se spojení s danou službou. Toto spojení se vytváří už pro účet uživatele a jeho úložiště, je tedy vyžadován přístupový token, podle kterého si služba ověří totožnost uživatele. Tyto tokeny jsou uloženy v databázi. Dropbox využívá token s neomezenou platností a Google Drive si, pokud je token již prošlý, vytvoří nový, pomocí refresh tokenu, který je také uložen. Po úspěšném vytvoření spojení se zavolá patřičná metoda pro nahrání souboru do úložištěm, která po dokončení nahrání vrátí data o nahraném souboru, jako je například id a jméno souboru v úložišti. Po nahrání souboru se některá tyto data vezmou a uloží se do databáze. Můžou pak být využity pro identifikaci tohoto souboru. Mazání písniček pak funguje podobně, ale místo toho aby se nejdříve pracovalo se službou a až poté se měnil obsah databáze, tak tento proces probíhá naopak. Smažou se data o souboru z databáze. Potom se vytvoří spojení se službou, pomocí uloženého tokenu uživatele a zavolá se metoda pro smazání souboru. V této metodě specifikujeme jaký soubor chceme smazat a to pomocí dat, které byly uloženy při nahrání tohoto souboru. Tyto data sice už v databázi nejsou, ale stále jsou načteny a uloženy v probíhající funkci. Po dokončení je soubor smazán jak z databáze, tak z úložiště.

Pokud je cílem těchto operaci Dropbox, tak se ke konci každé operace ještě zvlášť provádí dotaz na nový kurzor, který se pak uloží do databáze. To kvůli tomu, aby byla zaznamenána skutečnost,

že tyto změny, tedy uložení nebo smazání souboru, se již odráží v aktuálním stavu databáze.

4.2 Chytré hodinky

K vývoji aplikace na hodinky se použil Connect IQ SDK a vývojové prostředí Eclipse. Eclipse je open-source od společnosti Eclipse Foundation. Je hlavně využíváno pro programování v jazyce Java, ale má také spoustu rozšíření pro vývoj v jiných technologiích. Jedno z těchto rozšíření je pro vývoj aplikací pomocí Connect IQ. Toto rozšíření tento vývoj ulehčuje díky doplňujícím nástrojům, jako je například nástroj pro vyexportování aplikace, spouštění Connect IQ simulátoru nebo snadnější manipulaci s Connect IQ SDK Manager, který je tak možné otevírat rovnou z Eclipse. Connect IQ SDK Manager je ale nutno nejdříve stáhnout z oficiálních stránek výrobce. Na těchto stránkách je přímo popsaný postup pro nainstalování Connect IQ SDK Manager a nastavení prostředí Eclipse. Mimo Eclipse se s Connect IQ SDK dá pracovat pomocí příkazového řádku. [14]

Aplikace na hodinkách používá backend webové aplikace k synchronizaci hudebního obsahu. Pro identifikaci uživatele se používají JWT stejně jako u webové aplikace. Uživatel se přihlásí přes mobilní telefon a dostane zpátky JWT, který se uloží v hodinkách. Tento token se pak dále posílá při ostatních požadavcích k ověření uživatele. Životnost tohoto tokenu je ale značně prodloužena, aby se uživatel nemusel přihlašovat opakovaně. Po přihlášení si uživatel může synchronizovat hudební obsah, který se odvíjí podle toho, jaké úložiště má uživatel připojené a jakými písničkami jsou naplněné.

Pro synchronizaci slouží třída, která dědí ze třídy SyncDelegate. Ta má jednu pro nás dvě důležité metody a to onStartSync a isSyncNeeded. Metoda isSyncNeeded se zavolá ve chvíli když má začít synchronizace a vrací boolean hodnotu, jestli se má aplikace synchronizovat. Pokud vrátí true, tak se spouští metoda onStartSync, v opačném případe synchronizace nezačne. Když synchronizace začne, tak hodinky pošlou HTTP požadavek s JWT v hlavičce Authorization na backend. Ten ověří platnost JWT a případě úspěchu vrátí seznam playlistů a písniček od uživatele. Protože mají hodinky značně omezenou kapacitu úložiště, tak je nutné šetřit místem. Proto má seznam playlistů a písniček jiný formát, než kdyby se odesílal na frontend. Jsou vynechány údaje, které jsou pro hodinky nepodstatné.

```
{
    "songs"=>[
        {"name"=>"pisnicka1.mp3", "id"=>10},
        {"name"=>"pisnicka2.mp3", "id"=>20},
        {"name"=>"pisnicka3.mp3", "id"=>30}
    ],
    "playlists"=>[
        {"name"=>"playlist1", "id"=>1, "sync"=>false, "songsIds"=>[10,20,30]},
```

```
        {"name"=>"playlist2", "id"=>2, "sync"=>true, "songsIds"=>[100,200,300]}
    ]
}
```

Listing 4.1: Ukázka dat vrácených ze synchronizace

Poté se porovná obsah uložený v hodinkách s obsahem stáhnutým a zjistí se, které písničky je potřeba smazat nebo stáhnout a nový stáhnutý obsah se uloží do úložiště na hodinkách. Písničky se poté synchronně stahují a ukládají se. Abychom si udrželi na konkrétní stáhnuté písně referenci, tak uložíme do úložiště identifikátor, který samy hodinky vytvoří. Tyto identifikátory k písním jsou uloženy ve slovníku, kde klíčem je id písně v databázi a hodnotou vytvořený identifikátor.

```
{
    "10"=> 1648,
    "20"=> 1649,
    "30"=> 1650
}
```

Listing 4.2: Ukázka slovníku s uloženými identifikátory

Uživatel si ale může vybrat, které playlisty chce stáhnout a které nikoliv. Toto si může nastavit v checkbox menu v módu nastavení přehrávání. Menu zobrazí playlisty, které jsou uloženy v hodinkách bez ohledu na to, jestli jsou stáhnuté i jejich písničky. Uživatel potom vybere, které playlisty si přeje synchronizovat. Po potvrzení jeho volby se pak pošle HTTP požadavek na backend, který změní sync hodnotu u playlistu, podle toho jestli se má synchronizovat nebo ne a při další synchronizaci hodinek, se podle hodnoty sync ze stáhnutých dat stáhnou případné chybějící písně do těchto playlistů.

K přehrávání hudby pak slouží třída, která dědí z ContentIterator. Ta ve svých metodách vrací reference na písničky uložené v hodinkách, které získává díky uloženému slovníku s identifikátory těchto písniček. Playlist, který se má přehrát, si uživatel vybere v módu nastavení přehrávání. Jeho id se pak uloží do hodinkového úložiště a tato třída sestaví pole písniček pro přehrání, podle id písniček které playlist obsahuje. Pokud není žádný playlist zvolen, tak se vyberou k přehrání všechny písničky.

V úložišti hodinek je pak ještě uloženo nastavení vzhledu hudebního přehrávače, který může uživatel změnit v nastavení přehrávání. To se poté odráží na množství a typu tlačítek, které jsou zobrazeny v módu přehrávání. Při přehrávání hodinky ukazují aktuální skladku, která se přehrává, a informace o ní. Tyto informace čerpá z meta dat, které obsahuje mp3 soubor písničky.

V případě odhlášení uživatele jsou smazány všechny data uložené v hodinkách. Poté se může přihlásit jiný uživatel. Odhlášení je také možné využít k případnému vyresetování aplikace, protože

se smažou všechna data. Pak je možné opět se přihlásit a od začátku synchronizovat data na hodinkách.

Kapitola 5

Závěr

Cíl této bakalářské práce byl navrhnout a implementovat hudební aplikaci pro chytré hodinky Garmin a webovou aplikaci. Webová aplikace byla vytvořena za pomoci technologie .NET a Angular. Zase aplikace pro chytré hodinky byla postavena na technologii Connect IQ.

Zadané cíle byly splněny. Aplikace byla tvořena takovým způsobem, aby byla do budoucna rozšířitelná a mohla být doplněna o nové prvky nebo funkce. To platí jak to aplikaci webovou, tak pro aplikaci pro chytré hodinky Garmin.

Webová aplikace byla i úspěšně nasazena jako cloudová aplikace na Microsoft Azure a bylo jí tak možné otestovat jako kdyby byla plnohodnotně vydána. Aplikace na chytré hodinky byla zase publikována do Connect IQ obchodu s aplikacemi a nainstalována na fyzický model hodinek. Obě takto vydané aplikace se ukázaly jako funkční a nenastaly s nimi za ostrého provozu neřešitelné problémy.

Literatura

1. *App Types* [online] [cit. 2021-02-02]. Dostupné z: https://developer.garmin.com/connect-iq/user-experience-guidelines/app-types/.
2. *Duck-typing* [online]. San Francisco (CA): Wikimedia Foundation, 2021 [cit. 2021-02-02]. Dostupné z: https://cs.wikipedia.org/w/index.php?title=Duck-typing%5C&oldid=19347666.
3. *Monkey C Language Reference* [online] [cit. 2021-02-02]. Dostupné z: https://developer.garmin.com/connect-iq/reference-guides/monkey-c-reference/.
4. *Webová aplikace* [online]. San Francisco (CA): Wikimedia Foundation, 2019 [cit. 2021-02-02]. Dostupné z: https://cs.wikipedia.org/w/index.php?title=Webov%5C%C3%5C%A1_aplikace%5C&oldid=17371513.
5. *Introduction to Angular concepts* [online] [cit. 2021-02-02]. Dostupné z: https://angular.io/guide/architecture.
6. *Introduction to components and templates* [online] [cit. 2021-02-02]. Dostupné z: https://angular.io/guide/architecture-components.
7. *Introduction to modules* [online] [cit. 2021-02-03]. Dostupné z: https://angular.io/guide/architecture-modules.
8. *What is .NET?* [Online] [cit. 2021-02-03]. Dostupné z: https://dotnet.microsoft.com/learn/dotnet/what-is-dotnet.
9. WAGNER, Bill; NEWCOMB, Ben; WARREN, Genevieve; PINE, David; SCHONNING, Nick; ADDIE, Scott; KULIKOV, Petr; WENZEL, Maira; VICTOR, Youssef; LATHAM, Luke; ONDERKA, Petr. *A tour of the C# language* [online]. 2021 [cit. 2021-02-25]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/.
10. *How do I create an Audio Content Provider?* [Online] [cit. 2021-03-21]. Dostupné z: https://developer.garmin.com/connect-iq/connect-iq-faq/how-do-i-create-an-audio-content-provider/.
11. *Introduction to JSON Web Tokens* [online] [cit. 2021-03-03]. Dostupné z: https://jwt.io/introduction.

12. *Dependency Inversion Principle* [online] [cit. 2021-03-03]. Dostupné z: `https://www.tutorialsteacher.com/ioc/dependency-inversion-principle`.

13. *OAuth Guide* [online] [cit. 2021-03-10]. Dostupné z: `https://www.dropbox.com/lp/developers/reference/oauth-guide`.

14. *Getting Started* [online] [cit. 2021-04-03]. Dostupné z: `https://developer.garmin.com/connect-iq/connect-iq-basics/getting-started/`.

15. JEPSON, Brian. *Wearable Programming for the Active Lifestyle*. First edition. Sebastopol: O'Reilly Media, 2017. ISBN 9781491972090.

16. PRICE, Mark J. *C# 9 and .NET 5 – Modern Cross-Platform Development: Build intelligent apps, websites, and services with Blazor, ASP.NET Core, and Entity Framework Core using Visual Studio Code*. 5th Edition. Packt Publishing, 2020. ISBN 9781800568105.

17. SANCTIS, Valerio De. *ASP.NET Core 3 and Angular 9: Full stack web development with .NET Core 3.1 and Angular 9*. 3rd Edition. Packt Publishing, 2020. ISBN 1789612160.